the AARDVARK JOURNAL

VOLUME 3 NO. 3 AUGUST 1982

INDEX

RELOCATING WP-6502 PART II......1-2
VIDEO PATCH FOR 65D.............3
SOUND PROGRAM, G. ARTMAN........3-4
COMMAND MODIFICATIONS 65D.......4-6
DISK COMPRESSOR.................7-8
SCREEN PRINT J. SEYBOLD.........8
LETTER; DON VANSYCKEL...........9
LOCATING BAD 2114'S W. MCRICE...9-11
STRUCTURED PROGRAMMING..........11-13
PARALLEL PRINTER INTERFACE MOD..14
CLUTTER FOR OSI CORRECTION......14-15
FLAGG TOGGLE; JIM WEIMER........15
BASIC BLOCK DELETE CORRECTION...15
ODDS OR EVEN, V. BARBOUR........15
QUICK PRINTER OUTPUT;FLEISHMAN..16
ANIMAL GUESS GAME II............16-17
GALAXIA ADDITION................17
ALIEN II FOR C1E................17
PING PONG GAME..................17-18
POOL GAME.......................18-20

ATTENTION GRIFFIN FAMILY:

Last months article on 'The Hand Assembly of Programs for OSI's Machine Language Monitor' was written by none other than your very own GRANT GRIFFIN. Now are you impressed? You should be, it was a very good article. Keep up the Good Work Grant.

REQUEST TO JOURNAL AUTHORS:

Your cooperation is greatly needed if you have a printer and are sending in articles to be published in the Journal. We print letters, articles, listings etc. to 40 wide and many of you are sending them in at 70 wide. If at all possible when sending them in please print at 40 wide. Also I sometimes make mistakes trying to read some of the listings sent in when retyping them. Putting them on cassette may save some aggrivating error correcting in a good program. With them on cassette, they can be printed from the computer, error free. Thank you for your support. Cyndi

RELOCATION WP-6502 (PART 2)

BY John T. Roecker

I was overjoyed after purchasing an Epson MX-80 printer about two weeks ago. I had relocated WP-6502 in order to use it with my C1P with a C1S Monitor ROM. I knew WP-6502 was working because I could create a tape of an article or letter and take the tape to a friend who had a printer to have it printed. This was inconvenient and also could possibly tax a friendship. I had the RS232 interface populated so all I had to do was connect the Epson to it. I ran a quick test in BASIC and the printer worked fine. However, when I attempted to use WP-6502, the printer did not! After much head scratching I remembered that I had to use a different output routine in my C1S Monitor ROM in order to output to tapes using WP-6502. I surmized I would have to use this routine to output to the printer also Those of you who have C1E/C2Es can rest easy because these ROMs appear to use the standard output locations. A quick test checked my idea, the printer would work with WP-6502 with another change. A quick phone call to Rodger Olsen reinforced the need for a follow-up article.

My modification to WP-6502 was to add a new command, the Print command, to the WP-6502 repertoire and to disable using the View command to output to the printer. Those of you with standard OSIs or with the C1E/C2E monitor ROMs may find this new command useful.

I used the following steps to add the Print command to WP-6502. All address locations mentioned are the original addresses from your nonrelocated version of WP-6502. All instructions with an * behind them will have to have their address fields modified to suit your relocations.

1. Expand the WP-6502 menu so that the Print command may be added. The menu plus other words outputted to the screen are located at memory locations $070D through $0783. I modified the menu to have it look like this:

```
--WP
6502
Type
View
Blk View
G/Edit
L/Edit
Move
Print
Zap
R/Tape
W/Tape
```

I used the OSI Extended Monitor to relocate locations $0736 through the end of the cold start code of WP-6502, $0FD0, by 5 bytes. Then I added the following data at $0736:

```
$0732 4D6F76E5 Move
$0736 5072696EF4 Print
$073B 5A61F0 Zap
```

2. Contract the View command code the elminate the Pr? after View. The View command code is located at memory locations $0795 through $09F9. I dropped the instructions located at $0798, $079A, $079D, and $079F by relocating $07A1 to $0798.

3. In the process of performing these two relocations, I managed to destroy two instructions. One of these stopped L/Edit and G/Edit from working from the menu. The instruction which was destroyed for this problem was located at $078F. It should have the amount of your relocations added and subtracted from its address field. $078F 20940A JSR $0A94 *

The second instruction which was destroyed caused an insert at the End of Text to operate improperly. This instruction should be: $0C46 4C5A0F JMP $0F5A *

4. Any references which index into the WP-6502 menu may have to be corrected because we added a new command. References for commands after the new Print command will have to have 5 bytes added to the immediate data:

```
$03FA A052 LDY #$52
$0465 A040 LDY #$40
$06AD A05B LDY #$5B
$0787 A055 LDY #$55
$07B4 A043 LDY #$43
$07BB C04E CPY #$4E
$07E2 A050 LDY #$50
$09B0 A06D LDY #$6D
$09F0 A06D LDY #$6D
$0A5B A060 LDY #$60
$0B12 A03E LDY #$3E
$0B4D A071 LDY #$71
$0D01 A060 LDY #$60
$0D0C A060 LDY #$60
$0D44 A066 LDY #$66
$0E19 A059 LDY #$59
$0EA9 A06D LDY #$6D
```

5. The warm start code will have to be modified so it will recognize the Print command. Those of us with C1S/C2S monitor ROMs will have to add this check after the switch to the new output routine which was added in the last article.

This is what the code looks like before the change:

```
$0F8F E057 CPX #$57 Check for W/Tape
$0F91 D003 BNE $0F96
$0F93 20F30E JSR $0EF3 *
$0F96 4C6504 JMP  $0465  *  Not legal
command
```

This is after the change:

```
$0F8F E057       CPX  #$57   Check for
W/Tape
$0F91 D003     BNE $0F96
$0F93 20F30E   JSR $0EF3 *
$0F96 E050     CPX #$50  Check for Print
$0F98 D003     BNE $0F9D
$0F9A 4C9807   JMP $0798 * Print output
$0F9D  4C6504    JMP  $0465  *  Not legal
command
$0FA0 5D00
$0FA2  40                     Starting text
location
```

6. The cold start code will have to be modified to use the proper data/text starting location. I have indicated where this is in step 5 above; in this case $0FA2. The amounts of the relocations will have to modify this address. I have reproduced all the cold start code below. I placed my cold start code at $1024.

```
$1024 A94C      LDA #$4C     Store
$1026 8500      STA $00      warm
$1028 A90F      LDA #$0F     start
$102A 8502      STA $02      jump
$102C A90B      LDA #$0B     instruction
$102E 8501      STA $01      ----
$1030 A924      LDA #$24     Store
$1032 8503      STA $03      cold start
$1034 A910      LDA #$10     address
$1036 8504      STA $04      ----
$1038 A90F      LDA #$0F     Store
$103A 8D4202    STA $0242 *  starting text
$103D A9A2      LDA #$A2     address
$103F 8D4102    STA $0241 *  ----
$1042 A900      LDA #$00
$1044 8546      STA $46      ----
$1046 ADE2FF    LDA $FFE2    Test for C1P
$1049 D00A      BNE $1055    Branch if not
$104B A914      LDA #$14
$104D 8D3602              *
$1050 A9FF      LDA #$FF
$1052 8D4002    STA $0240 *
$1055 4C0000    JMP $0000    Jump to warm start
```

The immediate data in the instructions located at $1028 and $102C will have to be modified to suit your relocations. The immediate data in the instructions located at $1030 and $1034 point to the cold start code.

The immediate data at $1039 and $103E will have to be modified to point to your starting text address.

Now, after much blood, sweat, and some tears those of us with nonstandard monitor ROMs installed may use WP-6502 . To eliminate all this work, all machine language/Assembler code should start at a suitably high address. The Assembler/Editor starts at location $0240. I feel this would be a good starting address. Then anyone with standard or nonstandard monitor ROMs may use your program.

I have made additions to WP-6502 to utilize some of the features of my C1S monitor ROM. These additions will be the third article in this series.

RON WHITTAKER, SALT LK CITY, UTAH

Several weeks ago I wrote to R. Olsen describing the way I have modified my C1P so that the Video Mod II and the disk system both work. I also mentioned that I have the OS65D V3.3 operating system and that I had not yet been able to modify it to make use of the Video Mod II's wider display.

Since then, I have spent many hours trying to unravel the inner workings of the operating system. At last I have been successful! The patch to the OS is extremely simple to perform, even though I still don't understand it completely.

It seems there is a block of data bytes which starts at $32D5. Part of this block is swapped with page zero locations E0-F7 which is then used to control size and location parameters of the video display. Only three of these locations needed changing to alter the basic display. The byte at $32E9 is set at $17 by OS65D and is the number of lines in the display. The byte at $32EA is set at $17 also and is the number of characters per line. The byte at $32EB is the lower order half of the location of the upper left corner of the display and is set at $65. ($D0 is in $32EC. This is the high order half of this address but does not need to be changed.) If these three locations are changed, the video display is also changed. However, if any of the PRINT! commands are used, either from within a program or from the keyboard, the display reverts back to the original size. These PRINT! commands are new to OS65D V3.3 and control cursor location and movement and invoke special screen and line clear functions. I found three more data locations, $32F2, $32F3, and $32F4 which correspond to the same parameters as the first three; i.e. number of lines, characters per line, and upper left corner, respectively. These latter three locations are not swapped out to page zero but do control the PRINT! commands. By POKEing new values to these locations and then SAVEing these locations on disk, the new display parameters become a permanent part of the operating system and will be automatically loaded each time the disk is booted. There is one word of caution! All six POKES must be performed at the same time with no video accessing in between. Otherwise, the changes may be modified by the video access before they are completed. The following BASIC program will make the necessary changes to the six locations and then save the changes on the disk. RUN the program with each diskette that contains OS65D V3.3 and each will be changed.

```
10 PRINT!(28):REM...CLEAR SCREEN...
20 REM...POKE CHANGES...
30
POKE13033,27:POKE13034,31:POKE13035,64
40
POKE13042,27:POKE13043,31:POKE13044,64
50 REM...SAVE CHANGES TO DISK...
60 DISK!"SAVE 13,1=3274/8
```

The values I've POKEd into the six locations provide the maximum display size for my monitor which is a TV modified for direct video input. Other values may be needed for other systems.

MUSIC FOR C1P

For those of you who couldn't get Gerald Artman's Music program to work I think I may have the solution. I misplaced the cassette and didn't run off a listing. Sorry about that Jerry! Well here it is. (see June 82 issue for the rest of the article if you missed it.) Hope you enjoy it.

```
 2 REM**********GERALD D. ARTMAN JR.*****
*********
 3 REM**********3135  EVERGREEN  DR.*****
*********
 4 REM**********ROYAL OAK,MI.  48073*****
*********
 5 REM   Delete rems,
 6 REM   type only 5-10 numbered stateme
nts.
 9 REM SET UP NOTE ARRAYS, F=FREQ IN Hz
 10 DIMN$(18),F(18)
 15 FORX=1TO18:READN$(X),F(X):NEXT
 20 DATAC,261.62,C#,277.18,DF,277.18,D,2
93.66
 25 DATAD#,311.13,EF,311.13,E,329.63,F,3
49.23
 30 DATAF#,369.99,GF,369.99,G,391.99,G#,
415.30
 35 DATAAF,415.30,A,440,A#,466.16,BF,466
.16
 40 DATAB,493.88,0,15.15
 45 GOSUB305 :GOTO190
 49 REM INPUT NOTES ROUTINE
 50 PRINT:PRINT:INPUT"NEW SONG OR ADD";A
$:IFA$="N"THENX=1:GOTO60
 55 X=NS
 60 INPUT"TEMPO-BEATS PER MINUTE";T:PRIN
T:PRINT
 65 INPUT"NOTE BEAT VALUE (0.0)";B:IFB=<
0THENPRINT"INVALID":GOTO65
 69 REM OCTAVE 1 IS C BELOW BASS CLEFT
 70 INPUT"OCTAVE (1-5)";O:IFO>5ORO<1THEN
PRINT"INVALID":GOTO70
 75 INPUT"NOTE NAME";NN$:IFNN$="END"THEN
160
 80 FORI=1TO18:IFNN$=N$(I)THEN90
 85 NEXT:PRINT"INVALID":GOTO75
 90 FS=F(I):IFO=1THENFS=FS/4
 95 IFO=4THENFS=FS*2
 100 IFO=5THENFS=FS*4
 105 IFO=2THENFS=FS/2
 110 DU=((B*60)/T)*FS
 115 DH=INT(DU/256)
 120 DL=INT(DU-DH*256)
 125 DH=DH+1
 130 FQ=(((1000000/FS)-45)/70)
 135 F1=FQ-INT(FQ):IFF1>.5THENFQ=INT(FQ)
+1
 140 FQ=INT(FQ)
 145 IFNN$="0"THENFQ=0
 149 REM IF CORRECTION POKE IN REPLACEME
NT
 150 IFETHENPOKE4096+C*3,DL:POKE4096+1+C
*3,DH:POKE4096+2+C*3,FQ:GOTO190
 154 REM POKE IN VALUE DIRECTLY
 155 POKE4095+X,DL:X=X+1:POKE4095+X,DH:X
=X+1:POKE4095+X,FQ:X=X+1:GOTO65
```

```
 159 REM PUT END OF NOTE MARKER
 160 NS=X:POKE4095+X,0:POKE4095+X+1,0:PO
KE4095+X+2,1:GOTO190
 164 REM PLAY ROUTINE
 165 POKE55296,17:PRINTCHR$(25)
 170 PRINT"HIT SHIFT TO START":WAIT57100
,254,254
 175 POKE11,122:POKE12,2:X=USR(X):POKE55
296,1:PRINTCHR$(25)
 180 PRINT:PRINT:INPUT"REPLAY";A$:IFA$="
Y"THEN165
 189 REM   MENU
 190 FORX=1TO30:PRINT:NEXT:PRINT:PRINT"O
PTIONS:":PRINT:PRINT
 195 PRINT"EDIT A NOTE":E=0
 200 PRINT"GET SONG FROM TAPE":PRINT"INP
UT NEW SONG":PRINT"SAVE SONG"
 205 PRINT"PLAY":PRINT:PRINT:INPUT"CHOIC
E";A$:IFA$="G"THEN235
 210 IFA$="I"THEN50
 215 IFA$="S"THEN275
 220 IFA$="P"THEN165
 225 IFA$="E"THENE=1:INPUT"NOTE NUMBER "
;C:C=C-1:GOTO60
 230 GOTO190
 234 REM GET ROUTINE FOR TAPE INPUT
 235 INPUT"NEW OR ADD TO CURRENT NOTES";
A$:IFA$="N"THENSN=1:GOTO245
 240 SN=NS
 245 PRINT"START RECORDER AND HIT SHIFT"
:WAIT57100,254,254:POKE515,255
 249 REM LOOK FOR HEADER
 250 INPUTX:IFX<>0THEN250
 255 INPUTX:IFX=0THEN255
 259 REM READ IN AND POKE DIRECTLY
 260 INPUTD:IFD=9999THEN270
 265 POKE4095+SN,D:SN=SN+1:GOTO260
 270 POKE515,0:NS=SN-3:GOTO190
 274 REM OUTPUT TO TAPE ROUTINE
 275 PRINT"WHEN READY TO RECORD HIT SHIF
T":WAIT57100,254,254
 279 REM PUT HEADER ON TAPE
 280 POKE517,255:FORX=1TO10:PRINT0:NEXT:
PRINT255:X=1
 284 REM GET VALUE SEE IF END MARKER
 285 D=PEEK(4095+X):X=X+1:IFDOR(PEEK(409
5+X))=0THEN295
 290 PRINT0:PRINT1:PRINT9999:PRINT:PRINT
:GOTO300
 295 PRINTD:GOTO285
 300 POKE517,0:GOTO190
 304 REM MACHINE LANGUAGE ROUTINE FOR MU
SIC
 305 READN,N2:FORK=NTON2:READQ:POKEK,Q:N
EXT
 310 Q=INT(N/256):POKE12,Q:POKE11,N-Q*25
6:RETURN
 315 DATA560,691
 320 DATA166,242,142,0,223,170,202,234,2
34,234,234,234,234,234,234,234
 325 DATA234,234,234,234,234,234,208,238
,240,0,240,0,240,0,240,0
 330 DATA240,0,162,0,142,0,223,170,202,2
34,234,234,234,234,234,234
 335 DATA234,234,234,234,234,234,234,234
,208,238,198,243,208,5,198,244
 340 DATA208,190,96,240,0,240,0,208,183,
234,169,0,133,240,169,16
 345 DATA133,241,169,255,133,242,160,0,1
77,240,133,243,200,177,240,133
 350 DATA244,200,177,240,208,2,133,242,2
01,1,240,22,32,48,2,200
 355 DATA24,152,101,240,133,240,144,2,23
0,241,162,255,202,208,253,76
 360 DATA130,2,96,229
 365 END
```

KENNETH BOOTH, BOSQUE FARMS, NEW MEXICO

Of all the BASIC commands, I cannot think of one as useless as the LET command. Its use is optional so I personally never use it and have never gotten into trouble because I didn't. What I would like to have is a BASIC command that would clear the screen for me. An instant machine code screen clear would be of much greater use to me, and probably to you, then the LET command. The new command, call it SCL, would be just like any other BASIC command in that it wouldn't require the preliminary setup that the USR(X) function requires. Just enter SCL and the screen is instantly cleared. And it would work in the direct mode or inside a program. My BASIC now has this command and your BASIC can too, but, if you feel that you cannot do without the LET command then read no further because we are about to do away with it. The modifications to BASIC to enable us to do this are very simple and we will also incorporate some modifications to BASIC's addition, subtraction and multiplication routines that enable BASIC to run between 10 and 30 percent faster!

How does BASIC know where to go when it encounters a command? Our BASIC, like most others, uses a look-up table to find the command and a dispatch table to know where to go once it has found the command. The look-up table is a sequential list of all the commands in our BASIC's inventory and the dispatch table is a sequential list of addresses of routines that service these commands. The position of the address pairs in the dispatch table will match the position of the commands in the look-up table. If BASIC found a match for the encountered command at position 8 in the look-up table it would go to the 8th address pair in the dispatch table (lo byte-hi byte form, naturally) add 1 to the lo byte and branch to that address. If BASIC doesn't find a match, program execution cannot continue so BASIC branches to an error handling routine and kicks out those error codes that we all dearly love. The above explanation is greatly simplified but with this information we will be able to remove the LET command from the look-up table and insert out SCL command and place the proper address to service the SCL in the dispatch table.

On disk based machines running under OS65D V3.2, the look-up table is located at $0284 and the dispatch table is located at $0200. If you have never seen a look-up table, take a minute to look at it. Boot the Extended Monitor, call Track 2 into memory starting at $4200 (!CA 4200=02,1). Disassemble the code starting at $4284 (Q4284). =Shift P). The EM will display the Hex value of that memory location. Enter Shift 2. The EM will print the contents of $4284 as an ASCII character. Hit the line feed and the EM will display the next memory location. Enter Shift 2 again. If you continue on in this fashion that bunch of junk will start to make sense because BASIC's look-up table for commands will start scrolling up the screen. END is first, FOR is second, and so on. LET will be the 8th command on the list. The address for the routine that services the LET command will be the 8th address pair in the dispatch table. When BASIC encounters the LET command it will branch to $09A6.

If you noticed that the ASCII value of the last letter of every command does not match the true ASCII value of that letter you may be puzzled. Bit 7 of the ASCII value of the last letter of every BASIC command is set. All basic has to do is count the number of times Bit 7 is set and it always knows where it is at in the look-up table.

The speed modifications incorporated into the program are not my own. The original idea and program came from Mr. John A. Sauter of the University of Michigan. His article appears in the May 1981 issue of BYTE magazine. In his article, Mr. Sauter describes the bugs in OSI BASIC's addition, subtraction and multiplication routines and he submits an article to fix them. I included his fix in my program because we need a place to stash our code for the screen clear. By tightening up the code in the arithmetic routines, Sauter gives us enough room for our screen clear without using any free RAM. A big TALLY-HO to Mr. Sauter for an excellent article--read it for all the details on the speed up.

Please refer to the BASIC program listing. Line 100 calls Track 4 off of disk and into memory so we can modify it. Lines 200-260 POKE in the speed fixes along with our code for the screen clear. Line 300 modifies a branch instruction because of the revised code. Lines 400-420 POKE in more revised code for the speed fix. Line 500 saves the modified code out to disk. Line 600 calls Track 02 off of disk and into memory so we can modify BASIC's look-up and dispatch tables. Lines 610-630 modify these tables. The LET command is removed from the look-uptable and SCL is inserted in its place making sure that BIT 7 of the ASCII value of the letter L is set. The address for the routine that services LET is removed and the address of our screen clear is inserted It starts at $1862. Line 700 saves the modified code back to disk.

The program as submitted functions as advertised but it would be a good idea to have a back-up operating system available before you run it. One mistake in a DATA statement will crash a disk and make it useless unless you have a back-up or facilities for copying.

```
10 REM----BASIC SPEED ENHANCER----
20 REM----BASIC SCREEN CLEAR----
30 REM----KENNETH D. BOOTH----
40 REM----875 FOXCROFT LOOP----
50 REM----BOSQUE FARMS, NEW MEXICO---
60 REM---505 869-3945---
90 REM--CALL TK 4 INTO MEMORY SO IT CAN BE MODIFIED
95 REM--$ 4200=16986 dec.
100 DISK!"CA 4200=04,1"
150 REM--NOW POKE IN CORRECTED CODE
160 REM--CORRECTIONS ARE AT $4854 THRU $4884
200 FORX=18516TO18564:READY:POKEX,Y:NEXT
210 DATA 118,2,118,3,118,4,104,106,200
220 DATA 208,232,24,96,234,162,216,169
230 DATA 208,133,254,160,0,132,253,169
240 DATA 32,145,253,200,208,251,230,254
250 DATA 228,254,208,245,96,234,234,234
260 DATA 234,234,234,234,234,234,234,234
280 REM--CORRECT BRANCH AT $4846 TO REFLECT
290 REM--OUR REVISED CODE. THIS IS AN IMPORTANT
295 REM--POKE. SYSTEM WILL LOCK WITHOUT IT.
300 POKE18502,24
350 REM--POKE IN MORE CORRECTED CODE
360 REM--CORRECTIONS ARE AT $4946 THRU $4954
400 FORX=18758TO18772:READY:POKEX,Y:NEXT
410 DATA 102,115,102,116,102,117,102,118
420 DATA 102,189,152,74,208,214,96
450 REM--SAVE REVISED CODE OUT TO DISK
500 DISK!"SA 04,1=4200/8
550 REM--CALL TK 02 INTO MEMORY SO WE
560 REM--CAN MODIFY BASIC'S LOOK-UP AND
570 REM--DISPATCH TABLES TO USE OUR
580 REM--SCREEN CLEAR ROUTINE.
600 DISK!"CA 4200=02,1"
610 FORX=17054TO17056:READY:POKEX,Y:NEXT
620 DATA 83,67,204
630 POKE16910,97:POKE16911,24
640 REM--SAVE CORRECTED CODE BACK OUT TO DISK
700 DISK!"SA 02,1=4200/8
```

Listing #1
Basic Program That Effects
The Changes Described in Text
and Listing #2 and Listing #3

OLD CODE

```
1845  B03C  BCS $1883
1847  48    PHA
1848  B501  LDA $01,X
184A  2980  AND #$80
184C  5601  LSR $01,X
184E  1501  ORA $01,X
1850  9501  STA $01,X
1852  2448  BIT $48
1854  A900  LDA #$00
1856  9002  BCC $185A
1858  A980  LDA #$80
185A  5602  LSR $02,X
185C  1502  ORA $02,X
185E  9502  STA $02,X
1860  A900  LDA #$00
1862  9002  BCC $1866
1864  A980  LDA #$80
1866  5603  LSR $03,X
1868  1503  ORA $03,X
186A  9503  STA $03,X
186C  A900  LDA #$00
186E  9002  BCC $1872
1870  A980  LDA #$80
1872  5604  LSR $04,X
1874  1504  ORA $04,X
1876  9504  STA $04,X
1878  68    PLA
1879  08    PHP
187A  4A    LSR A
187B  28    PLP
187C  9002  BCC $1880
187E  0980  ORA #$80
1880  C8    INY
1881  D0C4  BNE $1847
1883  18    CLC
1884  60    RTS
```

NEW CODE

```
  1845  B018  BCS $185F
  1847  48    PHA
  1848  B501  LDA $01,X
  184A  2980  AND #$80
  184C  5601  LSR $01,X
  184E  1501  ORA $01,X
  1850  9501  STA $01,X
  1852  2448  BIT $48
  1854  7602  ROR $02,X
  1856  7603  ROR $03,X
  1858  7604  ROR $04,X
  185A  68    PLA
  185B  6A    ROR A
  185C  C8    INY
  185D  D0E8  BNE $1847
  185F  18    CLC
  1860  60    RTS
  1861  EA    NOP
  1862  A2D8  LDX #$D8
S 1864  A9D0  LDA #$D0
C 1866  85FE  STA $FE
R 1868  A000  LDY #$00
E 186A  84FD  STY $FD
N 186C  A920  LDA #$20
  186E  91FD  STA ($FD),Y
C 1870  C8    INY
L 1871  D0FB  BNE $186E
E 1873  E6FE  INC $FE
A 1875  E4FE  CPX $FE
R 1877  D0F5  BNE $186E
  1879  60    RTS
  187A  EA    NOP
  187B  EA    NOP
  187C  EA    NOP
  187D  EA    NOP
  187E  EA    NOP
  187F  EA    NOP
  1880  EA    NOP
  1881  EA    NOP
  1882  EA    NOP
  1883  EA    NOP
  1884  EA    NOP
```

Listing #2
Listing of Code
Before and After
Corrections

OLD CODE

```
1946  A900  LDA #$00
1948  9002  BCC $194C
1941  A980  LDA #$80
194C  4673  LSR $73
194E  0573  ORA $73
1950  8573  STA $73
1952  A900  LDA #$00
1954  9002  BCC $1958
1956  A980  LDA #$80
1958  4674  LSR $74
195A  0574  ORA $74
195C  8574  STA $74
195E  A900  LDA #$00
1960  9002  BCC $1964
1962  A980  LDA #$80
1964  4675  LSR $75
1966  0575  ORA $75
1968  8575  STA $75
196A  A900  LDA #$00
196C  9002  BCC $1970
196E  A980  LDA #$80
1979  4676  LSR $76
1972  0576  ORA $76
1974  8576  STA $76
1976  A900  LDA #$00
1978  9002  BCC $197C
197A  A980  LDA #$80
197C  46BD  LSR $BD
197E  05BD  ORA $BD
1980  85BD  STA $BD
1982  98    TYA
1983  4A    LSR A
1984  D0A4  BNE $192A
1986  60    RTS
```

NEW CODE

```
1946  6673  ROR $73
1948  6674  ROR $74
194A  6675  ROR $75
194C  6676  ROR $76
194E  66BD  ROR $BD
1950  98    TYA
1951  4A    LSR A
1952  D0D6  BNE $192A
1954  60    RTS
1955  EA    NOP
 .
 .
 .
1986  EA    NOP
```

Listing #3
Listing of Code
Before and AFter
Corrections

COMPRESSER

by James B. Perkins

COMPRESSER is a program, written in Basic, to allow more efficient use of valuable mini-floppy storage space.

I have found that, after only three months as a disk user, I've acquired twenty or so partially filled mini-floppies with one or two track gaps scattered throughout due to file deletions. Usually, I can find room among the good files for a short program I happen to be working on but, eventually, a new floppy has to be initialized to accomodate a larger file.

COMPRESSER will elminate all of these unused track gaps by copying all of the named files in the Directory down to consecutive tracks starting at Track 1. The entries in the Directory are altered to reflect the new locations of your files. If there are no gaps, then no effective action is taken as the program runs to completion.

A word of caution here! If you have saved by track number that are not named entries in the directory, they may be overwritten during the process.

While the program is written to take advantage of the extensions to Basic under OS65DV3.3, there are some changes noted later on that could allow it's use under earlier versions.

I have used screen memory at $D200 thru $D3FF as a buffer for the directory partly, in an effort to save workspace memory but, also because its a lot more fun to watch the track number being shuffled during execution than to stare at a static screen display. (You ought to see how I celebrate the 4th of July.)

Even though the program does not formally open or close disk files, it does require one buffer (Device #6) be provided when the program is typed in, as this area is used as a buffer by the CALL and SAVE commands. If you wish to eliminate this buffer from the program to save a track in the program file, then COMPRESSER can be modified by pokeing to 133 and 134 to provide a headspace buffer at the end of BASIC workspace memory. Lines 110 and 130 should also be changed to use the address of this new buffer.

Line 400 thru 420 are the instructional messages.

Lines 530 thru 620 establish the size requirements of Array DY%, needed to hold the valid Directory entries and to load those entries into the array.

Lines 20 thru 46 are a subroutine to sort the entries by track number for program use only. The order of entries in the Directory will be unchanged after program execution.

Lines 640 thru 730 are the working meat of the program. This section locates unused tracks and calls the subroutine at 100 to copy higher tracks down. It also updates the Directory buffer as it goes.

Lines 731 thru the end write the Directory buffer back to track 12 and terminate the program.

The TRAP commands at Lines 100 and 160 are necessary to accomodate the possibility of a multi-sector track such as are encountered when saving machine language object programs or as used in the Directory itself.

Speaking of the Directory, if you use COMPRESSER on a data disk that has unused tracks below the Directory, the Directory will be moved right along with everything else, thus rendering all of your fiels inaccessable. Nice huh? While I've made no provision to protect the Directory, a line of coding at Line 655 such as:

```
655 IF DY%(Y,1)=12 THEN NT=13:GOTO730
```

Should do the trick. Since I am still the humble owner of a single disk system, I like to keep the whole operating system on each floppy and this condition has not effected me.

Back to the Trap commands, if your files are entirely in Basic, they are always whole track multiples and, as such you won't need to provision for multi-sector copying. You could, therefore, remove those commands and move your Directory buffer off the screen and then delete all references to V3.3 Extended print commands such as "PRINT! (to enable operation of COMPRESSER under V3.2.

The reason that Line 160 contains a GOTO700 instead of a RETURN is that use of the Trap command seems to clear the stack of all FOR/NEXT loops and RETURNS from subroutine. Nasty business!

The peeks in Line 120 yield the sector number and number of pages in that sector that the DISK!"CALL" picked out of the sector header on the disk. 9821 is also useful as it holds the track number read from the header. These are listed in the PEEK/POKE list included in the V3.3 documentation but the description of them is vague. Its surprising the things you can find using the Extended Monitors string search features and a few lucky guesses.

One further disclaimer, while I haven't lost any files yet, you might want to make back-up copies of your irreplaceable files before running COMPRESSER until your confidence is established.

```
 1 REM DISK COMPRESSER
 2 REM JAMES B. PERKINS
 3 REM 7267 FAIRWOOD DR.
 4 REM INDPLS IN 46256
 5 REM 317-849-9099
 6 REM 19 JULY 82
 10 GOTO400
 20 DIMSF%(NE),S%(NE)
 22 Z=1:GOSUB28:GOSUB40:GOSUB30
 24 Z=2:GOSUB28:GOSUB30
 26 Z=3:GOSUB28:GOSUB30
 28 FORX=1TONE:SF%(X)=DY%(X,Z):NEXTX:RET
URN
 30 FORX=1TONE:DY%(X,Z)=SF%(S%(X)):NEXTX
:RETURN
```

```
 40 FORX=1TONE:P=0:FORY=1TONE
 42 IFSF%(X)>SF%(Y)THENP=P+1
 44 IFSF%(X)=SF%(Y)ANDX=>YTHENP=P+1
 46 NEXTY:S%(P)=X:NEXTX:RETURN
 100 S=1:WAIT160
 105 S$=STR$(S):S$=RIGHT$(S$,LEN(S$)-1)
 110 DISK!"CA 3A7E="+RT$+","+S$
 120 P=PEEK(9823):S=PEEK(9822)
 125 P$=STR$(P):P$=RIGHT$(P$,LEN(P$)-1)
 130 DISK!"SA "+NT$+","+S$+"=3A7E/"+P$
 140 IFP<7THENS=S+1:GOTO105
 150 RETURN
 160 WAIT0:PRINT!(18):GOTO700
 400 PRINT!(20):PRINT:PRINT:PRINT:PRINT"
 COMPRESSER"
 402 PRINT:PRINT:PRINT"    WARNING!!":PRI
NT:PRINT" FILES NOT IN DIRECTORY"
 404 PRINT" MAY BE LOST":PRINT:PRINT
 406 FORT=1TO500:NEXT
 408 PRINT" INSERT MINIFLOPPY TO":PRINT"
 BE COMPRESSED INTO"
 410 PRINT" DRIVE A":POKE2888,0:POKE8722
,0
 412 INPUT" PRESS <CR>";G$
 420 PRINT!(20)
 510 DEFFNA(X)=X-INT(X/16)*16+INT(X/16)*
10
 520 DEFFNB(X)=X-INT(X/10)*10+INT(X/10)*
16
 530 DO=53760:DT=DO+505
 540 DISK!"CA D200=12,1":DISK!"CA D300=1
2,2":NE=0
 550 FORY=DOTODTSTEP8:IFPEEK(Y+7)<>0THEN
NE=NE+1
 560 NEXTY:DIMDY%(NE,3):DP=1:PRINT!(18)
 570 FORY=0TO504STEP8
 580 IFPEEK(Y+DO+7)=0THEN620
 590 DY%(DP,1)=PEEK(Y+DO+6):DY%(DP,2)=PE
EK(Y+DO+7)
 600 DY%(DP,3)=Y
 610 FORX=1TO2:DY%(DP,X)=FNA(DY%(DP,X)):
NEXTX
 615 DP=DP+1
 620 NEXTY
 622 PRINT" SORTING";!(18)
 630 GOSUB20:NT=1
 640 Y=1
 650 IFDY%(Y,1)=<NTTHENNT=DY%(Y,2)+1:GOT
O730
 660 FT=NT:RT%=DY%(Y,1)
 670 NT$=STR$(NT):NT$=RIGHT$(NT$,LEN(NT$
)-1)
 680 RT$=STR$(RT%):RT$=RIGHT$(RT$,LEN(RT
$)-1)
 690 GOSUB100
 700 NT=NT+1:RT%=RT%+1:IFRT%<=DY%(Y,2)TH
EN670
 710 LT=NT-1:DP=DY%(Y,3):FT=FNB(FT):LT=F
NB(LT)
 720 POKEDO+DP+6,FT:POKEDO+DP+7,LT
 730 Y=Y+1:IFY<NE+1THEN650
 731 DISK!"SA 12,1=D200/1":DISK!"SA 12,2
=D300/1
 740 PRINT!(20):PRINT:PRINT" DISK COMPRE
SS":PRINT" COMPLETE"
 750 PRINT" NEXT AVAILABLE":PRINT" TRACK
 IS";NT:PRINT:PRINT
 760 INPUT"  <CR> TO CONTINUE";G$
 770 POKE2888,27:POKE8722,27:RUN"BEXEC*"
```

JOHN SEYBOLD, FULLERTON,CALIFORNIA

I want to thank Mr. Price for that wonderful screen print program that he had in the April issue of the Journal. It works very nicely for printing plots that you have on the screen, provided that you use characters that your printer will recognize of course. Like most of Journal readers though, I seldom leave well enough alone. Below is a modified listing of Mr. Price's program.

The most significant modification that I made was to adjust it to print 26 instead of 21 lines. This enables the program to print any headings or subtitles that you might have. I then went on to put multiple statements per line to reduce memory use. As can be seen from the sample run, you can now get the entire visible screen printed. The final change that I made was to reverse the logic in line 50, for deciding when to stop reading the screen, to save a few extra bytes by eliminating one goto.

```
1 REM  SCREEN PRINT JOHN PRICE KNOXVILLE
IA
2 REM  PROGRAM SCANS  VIDEO  MEMORY
LOCATIONS THEN PRINTS
3 REM THOSE CONTENTS TO A LINE PRINTER
4 REM REVISED  BY JOHN S SEYBOLD
FULLERTON CA
5 REM AARDVARK VOL 3 NO 1 PAGE 9
10 DIMS(24,26):C=53381:D=53405:X=0:Y=1
20 FORA=CTOD:X=X+1
30 Z=PEEK(A):S(X,Y)=Z:IFX=24GOTO50
40 NEXTA
50
Y=Y+1:C=C+32:D=D+32:X=0:IFD<54174GOTO20
60 SAVE:FORY=1TO26:FORX=1TO24
70 PRINTCHR$(S(X,Y));:IFX=24THENPRINT
80 NEXTX:NEXTY:POKE517,0:END

FORI=0TO20:?TAB(I);I:NEXT:RUN
0
 1
  2
   3
    4
     5
      6
       7
        8
         9
          10
           11
            12
             13
              14
               15
                16
                 17
                  18
                   19
                    20
```

DON VANSYCKEL, MIDDLEBURY, VERMONT

I am running C8P with OS65D V3.2 and have been experiencing a peculiar problem. One program which I wrote uses several arrays dimensioned at 58. Occasionally, the sytem would get lost and forget where the arrays were located. In fact, a print of element #1 in the BASIC immediate mode would return a bad subscript (BS) error. The system must have found the array name or I would think it would have generated the array again and found the element.

Time passes and I made a few minor changes to the program. Now, occasionally the last defined arithmetic variable will get clobbered. Some of the arrays are integer arrays but most are string arrays. I'll also mention here that at no time have I ever gotten an out of memory (OM) error. Also, the glitch happens at various points in loadin the arrays. I might get half the data in and the system will glitch. I'll then restart the BASIC program and the whole thing runs fine. The only difference in the two runs of the program is in data entry. As each block of data is entered it is displayed and corrections may be made to individual items. Between the two runs the number of and/or size of intermediate strings was different.

Time passes and I got the idea to delete some BASIC code, utility package, to give the variables more room. Since that time I have not experienced the glitch again. The problem is that I woud like to add some more code to the program but have no idea how much is safe to add before the system will glitch.

I have calculated that variable memory is large enough to easily hold all the variables. Also, extensive memory diagnostics have been run several times. However, under some conditions I think the garbage collection routine writes a string over the last arithmetic variable and then one of two things happens.

1. The arithmetic variable is read and has a bad value stored in it.
2. The arithmetic variables has a new value written into it which overwrites the string header (which should not be there); then, when the string is called the system can't find the string.

Possibly one of your readers has solved this problem or can define it better so that it can be avoided. Hopefully someone has patches to BASIC which they'll share.

WAYNE MC RICE, SAN FRANCISCO CA

I recently added the 8K memory board from AARDVARK to my C1P. Presently, I'm using it without the PIA and I've got 4K on the board so far.

I've come to realize, as I expand the memory, the importance of being able to locate a bad 2114 should the need arise. The program for testing memory that's included in the data sheets for the board is, as stated, very reliable. But, also as stated, it is very slow.

Now if you're as addicted as I am to punching away at the keyboard and staring bleary eyed at the monitor for hours on end, then having the machine tied up for five and six hours to run a test is akin to going cold turkey.

Well, wait no more. Here's a program that's just as efficient but a little less time consuming. The testing is done at the machine code level, and only returns to BASIC if there's a bad memory location or the testing is completed. As written, it's set up to test as much memory as possible. So it starts at $0400, slightly behind the BASIC program, and runs to the end of RAM. In my case that's $2FFF. But I assumed that most would have their boards fully populated if they have as much patience as me, but a few more dollars. So it's written to test to the end of 16K($3FFF), and I've included the necessary changes for 12K, 8K and 4K.

When I run it in my machine from $0400 to $2FFF, it takes about three and 1/2 minutes with no exits for errors. So, it's over with before I start getting the shakes.

THE WAY IT WORKS

The object is to put a byte of data in each memory address in RAM then check that address to see if it's acutally holding the data just stored there.

In the first half of the test, Register "A" is loaded with the data then stored at the first memory address. The data just stored is then immediately compared to the data in "A". If they are the same, the memory address is incremented and the same thing is repeated at the new address. If they aren't the same, the program first changes the Lo Byte of the User pointer at $000B then exits to BASIC to print, in decimal, where the error is, what number was poked into the address, and what number was returned. Then it jumps back into the program via the User pointer, resets "A" to what it was, increments the memory address and continues. The "X" and "Y" registers are used to test for the end of RAM. They are also reset if the program exits for an error call. After each address has been checked with this byte of data, the data in "A" is incremented, the address reset to the beginning, and the entire process is repeated so that every location is tested with data from $00 to $FF.

The second half of the program is only slightly different. Register the addresses. After all locations are loaded, they are checked, one after the other, to see if they are still the same as "A". If so, the same happens as before. The User pointer is changed, the program exits to BASIC to tattle, then returns and continues. When the Hi Byte of the memory address being tested matches the data held in "X", the testing is complete. This is done for both halves of the program. When the match is made in the second half, the program terminates. I imagine about five minutes for 16K if it doesn't stop for errors.

If, for some reason, the program is run again immediately, certain addresses must be reset by hand by going into the Monitor. $002A, 2D, 76 and 89, which hold the Hi byte of the address being tested, will now contain $40 in the case of 16K. They have to be reset to $04. If there were any exits for errors, the Lo Byte of the User pointer will also need to be changed back to $22. So, with a maximum of five changes, it's ready to go again.

There are two BASIC programs here. One is the DATA program and the other is the MAIN program.

The first step is to type in the DATA program and RUN it. If you want to SAVE it on tape, type it in and SAVE it before RUNning. It has a NEW command in line 25, and after it runs, the data is left in page 2 but the BASIC program is cleared out. The DATA program also sets the User pointer at $000B, C.

The second step is to type in the MAIN BASIC program. When it's in just RUN it.

The other method for loading the program, and I believe the easiest, is to first put in the MAIN program, then go into the Monitor to $0222, and use the disassembly listing to load the data. Be sure to put $22 into $000B and $02 into $000C. Then Warm start and RUN.

If you want to SAVE the disassembly on tape and load it directly into the monitor for use and you don't know how to do this yet, get a copy of THE FIRST BOOK OF OSI. There's a lot of good stuff in there. It's worth far more than it's price.

O.K. So maybe you don't have 16K either. With a few quick additons or changes the program can be adjusted to whatever you like.

These are the changes for 12K, 8K, and 4K:

To the BASIC DATA program add line 22.

(FOR 12K)

```
22 POKE547,48:POKE613,48:POKE623,48:
POKE699,48
```

(FOR 8K)

```
22 POKE547,32:POKE613,32:...ETC.
```

(FOR 4K)

```
22 POKE547,16:...ETC.
```

In the MAIN BASIC program change lines 20 & 35

(FOR 12K)

```
20 IFPEEK(A)=48THEN...ETC.
35 IFPEEK(A)=48THEN...ETC.
```

(FOR 8K)

```
20 IFPEEK(A)=32THEN...ETC.
35 "          "=32"    ...ETC.
```

(FOR 4K)

```
20 IFPEEK(A)=16...
35 " " " " "=16...
```

To the DISASSEMBLY LISTING change these:

ADDRESS	12K	8K	4K
$0223	30	20	10
$0265	30	20	10
$026F	30	20	10
$02BB	30	20	10

If you've never had the misfortune of getting a bum 2114, that's great! On the other hand, if like me, you have, then you probably are aware of the agony of trying to isolate it. And if you were really unfortunate, and put 2 bad chips on your board at the same time.....Wellllll..........

Last, but of course not least, I'd like to express my appreciation to the people at AARDVARK for their efforts to shed some light on the mysteries of OSI. Without your work I fear there would be a lot more dark corners than there are.

I've learned more from the listings to your games and the data sheets and the Journal than I could have possibly have learned from any other source. I have to mention the FIRST BOOK OF OSI again here, also. For me, it's been a real treasure.

BASIC DATA PROGRAM

```
5  REM 16K MEMORY TEST DATA
10 FORX=546 TO 707
15 READA:POKEX,A:NEXT
20 POKE11,34:POKE12,2
25 PRINT"READY":NEW
30  DATA162,64,160,0,169,0,141,0,4,205,
0,4,208
32 DATA47,238,41,2,238,44,2,204,41,2
35  DATA240,3,76,40,2,238,42,2,238,45,2,
236,42,2,240,3,76,40,2,238,39,
40 DATA204,39,2,240,26,169,4,141,42,2,
141,45,2,76,38,2,162,100,134,11
45 DATA96,162,64,160,0,173,39,2,76,48,2,
162,64,160,0,169,0,141,0,4,23
50  DATA117,2,204,117,2,208,245,238,118,
2,236,118,2,208,237,205,0,4,20
55 DATA41,238,136,2,204,136,2,208,243,
238,137,2,236,137,2,208,235,238
60 DATA115,2,204,115,2,240,21,140,117,2,
140,136,2,169,4,141,118,2,141
65 DATA137,2,76,114,2,162,186,134,11,96
70 DATA162,64,160,0,173,115,2,76,140,2
```

MAIN BASIC PROGRAM

After LOADing and RUNning the data program, LOAD and RUN this BASIC program:

```
1  REM 16K MEMORY TEST
5  D=256
10 X=USR(X)
15 A=554:B=553:C=551
20 IFPEEK(A)=64THEN30
25 PRINT"ERROR AT "PEEK(A)*D+PEEK(B):
GOSUB45:GOTO10
30 A=649:B=648:C=627
35 IFPEEK(A)=64THENPRINT"DONE":STOP
40  PRINT"STATIC ERROR  AT  "PEEK(A)*D+
PEEK(B):GOSUB45:GOTO10
45 PRINT"POKED"PEEK(C)" GOT "PEEK(PEEK
(A)*D+PEEK(B)):RETURN
```

DISASSEMBLY LISTING for MEMORY TEST

```
0222 A2 40 *     024F CC 27 02    0277 EE 75 02    02A4 8C 75 02
0224 A0 00       0252 F0 1A       027A CC 75 02    02A7 8C 88 02
0226 A9 00       0254 A9 04       027D D0 F5       02AA A9 04
0228 8D 00 04    0256 8D 2A 02    027F EE 76 02    02AC 8D 76 02
022B CD 00 04    0259 8D 2D 02    0282 EC 76 02    02AF 8D 89 02
022E D0 2F       025C 4C 26 02    0285 D0 ED       02B2 4C 72 02
0230 EE 29 02    025F A2 64       0287 CD 00 04    02B5 A2 BA
0233 EE 2C 02    0261 86 0B       028A D0 29       02B7 86 0B
0236 CC 29 02    0263 60          028C EE 88 02    02B9 60
0239 F0 03       0264 A2 40 *     028F CC 88 02    02BA A2 40 *
023B 4C 28 02    0266 A0 00       0292 D0 F3       02BC A0 00
023E EE 2A 02    0268 AD 27 02    0294 EE 89 02    02BE AD 73 02
0241 EE 2D 02    026B 4C 30 02    0297 EC 89 02    02C1 4C 8C 02
0244 EC 2A 02    026E A2 40 *     029A D0 EB
0247 F0 03       0270 A0 00       029C EE 73 02    000B 22
0249 4C 28 02    0272 A9 00       029F CC 73 02    000C 02
024C EE 27 02    0274 8D 00 04    02A2 F0 15       *Change points for
                                                   other than 16K.
```

USING STRUCTURED PROGRAMMING TECHNIQUES

by Larry Ellenbecker

I have enjoyed your Journal very much over the past 2 years. In that time I have not seen any articles dealing with two very important areas of programming: 1) program documentation, and 2) using structured programming in developing BASIC programs. Since you mentioned re-establishing the Beginner Corner I felt this might be a good time to submit an article.

A common excuse among programmers is that they never have time to write user documentation for their programs. Particularly in a business environment and even for the serious hobbiest documentation of programs is very important. A business is concerned with retaining control over the programs it has paid to develop and depends upon to conduct its normal activities. Also for businesses, there is always the potential for staff turnover to consider. The personal computerist should be concerned with program documentation for other reasons. As the computerist develops his or her programming skills, documentatio of techniques that have been mastered can be invaluable when it comes time to program more sophisticated applications. Through the use of documentation, you can build your own reference library of computer programming techniques. By now the reader is saying "that sounds good, but documentation is a time consuming process". My answer to this is "it doesn't have to be that way if you use structured programming techniques".

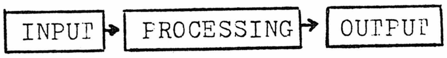
I recently had the opportunity to enroll in a COBOL programming course. During the entire course the instructor continually stressed the importance of structured programming in writing programs that could be easily maintained. Structured programming uses a top-down approach in programming. A modular structure chart is initially prepared which simply outlines the major processes that the program must take into account in order to accomplish a specific task. From the modular structure chart, a flow chart is developed which details each process as a separate operational portion of the program. The flow charted modules can then be used to write the source code program and go through the debugging process.

The instructor of my COBOL course imposed two interesting restrictions on us as students in his course. First, we were not allowed to use the COBOL command GO TO to branch from one operational process to another. We could only use GO TO to exit out of a processing loop. Second, we were required to use the PERFORM command to execute the various modular processes in our program. In BASIC this would be like using a GOSUB command. Well, to make a long story short, my COBOL instructor convinced me that structured programming is a valuable technique regardless of programming language. GOTO statements in BASIC programs are among the most overused and most sloppily used statements available to a programmer. Indiscriminate use of GOTO makes documentation writing very difficult and debugging next to impossible. I'm not saying that programmers should quit using GOTO's, but rather that GOSUB's and GOTO's can be used more effectively if you disipline yourself to think about your programs from a structured point of view.

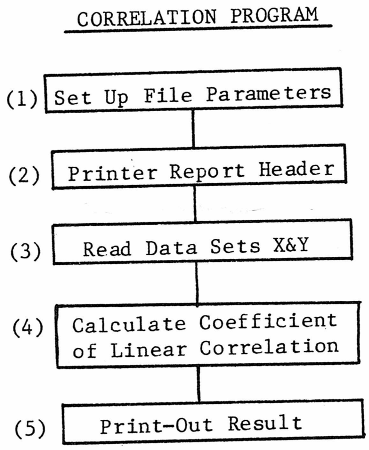
To demonstrate what I mean, I will go through an application programming process using structured programming concepts. The application we will be programming will be to calculate the coefficient of linear correlation.

Modular Structure Charts

In this phase of developing a program, we are concerned with conceptualizing the programming task. In their simplest form all processes can be depicted by the Input-Output Model.



However, if we really begin to think about a programming application we can usually break it down into component parts or processes. Our thoughts can be schematically charted as shown in the modular structure chart below:


CORRELATION PROGRAM
(1) Set Up File Parameters
(2) Printer Report Header
(3) Read Data Sets X&Y
(4) Calculate Coefficient of Linear Correlation
(5) Print-Out Result

Once we have drawn this chart you may ask; what does it tell us and how does it help to develop a flow chart and ultimtely a program. This will become obvious as we review each component of our sample structure chart. Item (1), Set Up File Parameters; is used for all initializing operations. For example, if we are dealing with arrays and matrixes, this is the part of the program where they are dimensioned. Counter values are also set at this point in the program. Item (2), Print Report Header; here we might be concerned with printing a hard copy of our calculation and would like to provide an appropriate title for the output. Item (3), Read Data; as in any data processing we must read or input the data from somewhere. Item (4), the Calculating Correlation Coefficients; is a mathematical process and therefore relies on a standard formula. Item (5), Print-Out as in any process we will output some kind of results.

Flow Charting

At this point, depending upon the complexity of the operations in each of these modular levels, we may flow chart some, all, or none of the defined modules. Because of space constraints in this article I will not show a flow chart for this program example. However, the reader should be aware that Items 3 and 4 are complex enough to warrant flow charting depending upon the programming abilities of the individual programmer.

Structured Programs

I would now direct the readers attention to the program listing for the "CORREL" program. This is an example of a structured BASIC program. Lines 10-90 provide the user with several items of useful information; such as, buffers required, system requirements and formula source.

Lines 100-190 constitute the structure for the operation of the entire program, hence the title "CONTROL MODULE". The Control Module also provides a potential user with complete operational documentation for the program logic because it specifically directs the user to the source code lines for each operation. Note the use of the GOSUB and GOTO statements. GOSUB's are used to perform all main modular activities while GOTO statements are used within modules to alter operations upon meeting the conditional requirements of a control break statement (see example of conditional IF statement in line 1020).

In addition to the documentation that is provided in the Control module, it is easy to document specific operations by referring to a particular module. For example, in the "Read Data Module" statement lines 1200 through 1280, lines 1230 and 1260 have a particular operation performed on the (X data set) and (Y data set) that are input for processing. The data that I processed with this program had numberic values with two decimal places for the (X data set) and one decimal place for the (Y data set). To save disk storage space, the data was entered without the decimal points. If I came back to use this program six months from now, I know I wouldn't remember that these lines were set up that way. However, if I document that fact as an element of the "Read Data Module", I have a reference to consult at some later date before program execution. A second example of where simple documentation can be helpful is in the "Print Report Header Module", lines 710 and 740. The REM statements are used to show where the printer could be turned on and off. If you want a hard copy of your calculation, simply insert 710 POKE8994,3 and 740 POKE8994,2. As you can see, useful documentation can be as simple as a single written statement directed to a specific program line number under a specific modular heading. The documentation page can be attached to the program source code and your structured program is complete.

In conclusion, I think this scussion about structured programming d documentation can serve personal and business computerists well if they remember some of the important steps in the process.

Structured Programming Techniques

1. Conceptualize your programming task and draw up a modular structure chart that includes all the important operational components.
2. Use flow charts to aid in defining difficult operations withing each modular component.
3. Discipline yourself in using GOSUB's and GOTO statements to maximize your control over programming processes.
4. Use the "Control Module" to document overall program logic; and then document specific operations by referring to the module name and appropriate program line number.

```
10 REM  BUFFER 6 & 7 ATTACHED
20 REM
30 REM  FILE NAME     :  CORREL
40 REM  WRITTEN BY    :  LARRY L. ELLENBECKER
50 REM  PGM DESC.     :  COEFFICIENT OF LINEAR CORRELATION PROGRAM
51 REM                   FORMULA SOURCE-(ADVANCED BASIC) BY
52 REM                   JAMES S. COAN
60 REM  SYSTEM        :  OSI C4PMF (MIN. SINGLE DISK DRIVE REQUIRED)
70 REM  SUPPORT PGMS  :  "DATA E"  DATA ENTRY PROGRAM
80 REM  DATA          :  4-2-82
90 REM
100 REM  CONTROL MODULE
110 GOSUB  500:REM  SET UP PARAMETERS/DIMENSIONS
115 GOSUB  700:REM  PRINT REPORT HEADER
120 GOSUB 1000:REM  OPEN DISK FILE BUFFER 6
130 GOSUB 1100:REM  OPEN DISK FILE BUFFER 7
140 GOSUB 1200:REM  READ DATA
    GOSUB 2000:REM  SUMMATION CALCULATED (X&Y DATA SETS)
    GOTO [redacted] 140
170 GOSUB 2500:REM  VARIANCES CALCULATED
180 GOSUB 3000:REM  PRINT OUT COEFFICIENT OF LINEAR CORRELATION
190 END
500 REM  SET UP PARAMETERS/DIMENSIONS
510 PRINTTAB(13)"COEFFICIENT OF LINEAR CORRELATION PGM"
520 X=0:Y=0
530 FOR Z=1 TO 35:PRINT:NEXT
540 PRINT:PRINT:PRINT:RETURN
700 REM  PRINT REPORT HEADER
710 POKE 8994,3
720 PRINTTAB(36)"THE COEFFICIENT OF LINEAR CORRELATION FOR DATA SETS X & Y"
730 PRINT:PRINT
740 POKE 8994,2
750 RETURN
1000 REM   OPEN DISK FILE BUFFER 6
1010 READ F1$:PRINT:PRINT"(X) DATA SET BEING READ"
1020 IF F1$="END" THEN GOTO 170
1030 DISK OPEN,6,F1$
1040 RETURN
1100 REM   OPEN DISK FILE BUFFER 7
1110 READ F2$:PRINT:PRINT"(Y) DATA SET BEING READ"
1120 DISK OPEN,7,F2$
1130 RETURN
1140 REM
1150 DATA "COR D1","COR D5","COR D2","COR D6","COR D3","COR D7"
1160 DATA "COR D4",COR D8"
1170 DATA "END"
1200 REM   READ DATA
1210 INPUT#6,IV$
1220 IF IV$="END-FILE" THEN GOSUB 1000:GOTO 1210
1230 XX=VAL(IV$):X=(XX/100)
1240 INPUT#7,IV$
1250 IF IV$="END-FILE" THEN GOSUB 1100:GOTO 1240
1260 Y=VAL(IV$)/10
1270 N=N+1
1280 RETURN
2000 REM  SUMMATIONS CALCULATED FROM X AND Y DATA SETS
2010 S=S+X*Y:S1=S1+X:S2=S2+Y:T1=T1+X↑2:T2=T2+Y↑2
2020 RETURN
2500 REM  VARIANCES CALCULATED
   0 A1=S1/N:A2=S2/N:B1=T1/N:B2=T2/N:V1=SQR(B1-A1↑2):V2=SQR(B2-A2↑2)
     R=(N*S-S1*S2)/((N↑2)*V1*V2)
     PRINT:PRINT"VARIANCES CALCULATED":PRINT
2540 RETURN
3000 REM  PRINT COEFFICIENT OF LINEAR CORRELATION
3010 FOR Z=1 TO 35:PRINT:NEXT
3020 POKE 8994,3
3030 PRINTTAB(40)"CORRELATION COEFFICIENT =";R:PRINT
3040 POKE 8994,2
3050 S=0:S1=0:S2=0:T1=0:T2=0
3060 RETURN

OK
```

The benefit you will receive for your efforts will be:

1. A program that is easy to alter should you wnat to expand its capabilities. Other subroutines can be added anywhere and their operation controlled by its sequence in the "Control Module" statements
2. Access to a reference base of mini programs (modules) that are already debugged and which can easily be adopted for use in other structured programs.

I have also included a copy of a program called "DATA E" for data entry for running the "CORREL" program. It is another example of a structured program which I hope will be adaptable for use by other computerists.

```
10 REM  BUFFER 6 ATTACHED
20 REM
30 REM  FILE NAME    :  DATA E
40 REM  WRITTEN BY   :  LARRY L. ELLENBECKER
50 REM  PGM DESC.    :  DATA ENTRY PROGRAM FOR (CORREL) PROGRAM
51 REM                  COEFFICIENT OF LINEAR CORRELATION PGM
60 REM  SYSTEM       :  OSI C4PMF (MIN. SINGLE DISK DRIVE REQUIRED)
70 REM  SUPPORT PGMS :  PROVIDES FORMATTED DATA FOR "CORREL" PGM
80 REM  DATA         :  4-2-82
90 REM
100 REM     CONTROL-MOD
110 GOSUB  500:REM  SET DIMENSIONS
120 GOSUB 1000:REM  FILE-OPEN-MOD
130 GOSUB 2000:REM  DATA-ENTRY-MOD
140 GOSUB 5000:REM  REVIEW & CORECT-MOD
150 GOSUB 6000:REM  DATA-TO-DISK-MOD
160 GOSUB 3000:REM  CLOSE-FILE-MOD
170 GOSUB 4000:REM  MANUAL-CONTROL-MOD
180 IF FLAG=1 THEN FLAG=0:GOTO 120
190 END
200 REM
500 REM    SET DIMENSIONS
510 POKE 2893,28:POKE 2894,11
520 DIM X$(16,25),S(25)
530 I=0:K=0
540 Q=16 ←      NOTE: Q Controls the size
                      of the DATA MATRIX
550 RETURN
1000 REM    FILE-OPEN-MOD
1005 FOR X=1 TO 35:PRINT:NEXT
1010 FLAG=0
1020 PRINT:PRINT:INPUT"ENTER FILE NAME";F$
1030 IF F$="END" GOTO 160
1040 DISK OPEN,6,F$
1080 FOR X=1TO35:PRINT:NEXT:RETURN
2000 REM      DATA-ENTRY-MOD
2010 PRINT"DATA ENTRY MODULE":PRINT:PRINT
2020 FOR I=1 TO Q
2030 FOR J=1 TO 25
2040 G=G+1:PRINTG;:INPUT"ENTER DATA:";D$:PRINT:PRINT
2050 IF D$="END-FILE" THEN GOTO 2090 ←
2060 X$(I,J)=D$                          NOTE: ENTER
2070 NEXT J                           "END-FILE" To Exit
2080 NEXT I                           DATA ENTRY Loop
2090 G=0:RETURN
3000 REM      CLOSE-FILE-MOD
3010 DISK CLOSE,6
3020 RETURN
4000 REM         MANUAL-CONTROL-MOD
4010 FOR X=1 TO 30:PRINT:NEXT
4020 INPUT"OPEN NEXT FILE ? -ENTER (Y/N)-";Y$
4030 IF Y$="Y" THEN FLAG = 1
4040 I=0:K=0
4050 RETURN
5000 REM  REVIEW & CORRECT-MOD
5010 PRINT:PRINT:PRINT:PRINT"REVIEW & CORRECT"
5020 FOR I=1 TO Q:PRINTTAB(18)"DATA SET ";I
5030 FOR J=1 TO 13
5040 PRINTJ;" = ";X$(I,J);TAB(32)J+12;" = ";X$(I,J+12)
5050 NEXT J
5060 GOSUB 7000
5070 IF K=0 THEN 5090
5080 FORM=1TOK:Y=S(M):PRINTX$(I,Y);:INPUT"CORRECTION:";C$:X$(I,Y)=C$:NEXT
5090 FOR X=1 TO 30:PRINT:NEXT:NEXT I:RETURN
6000 REM  DATA-TO-DISK-MOD
6010 FOR I=1 TO Q
6020 FOR J=1 TO 25
6030 PRINT#6,X$(I,J)
6040 NEXT J
6050 NEXT I
6070 TE$="END-FILE":PRINT#6,TE$
6080 PRINT:PRINT"DATA PRINTED TO DISK FILE";F$
6090 FOR X=1 TO 2000:NEXT:RETURN
7000 REM  ERROR-LIST-MOD
```

```
7005 K=0
7010 PRINT:INPUT"ENTER NUMBER OF ERRORS (0-n)";N
7020 IF N=0 THEN GOTO 7070
7030 PRINT:PRINT"ENTER SUBSCRIPT OF ERRONEOUS ENTRY"
7040 K=K+1:INPUT S(K)
7050 T=T+1:IF T=N THEN GOTO 7080
7060 GOTO 7030
7070 FOR X=1 TO 30:PRINT:NEXT:K=0
7080 PRINT:N=0:T=0:RETURN
```

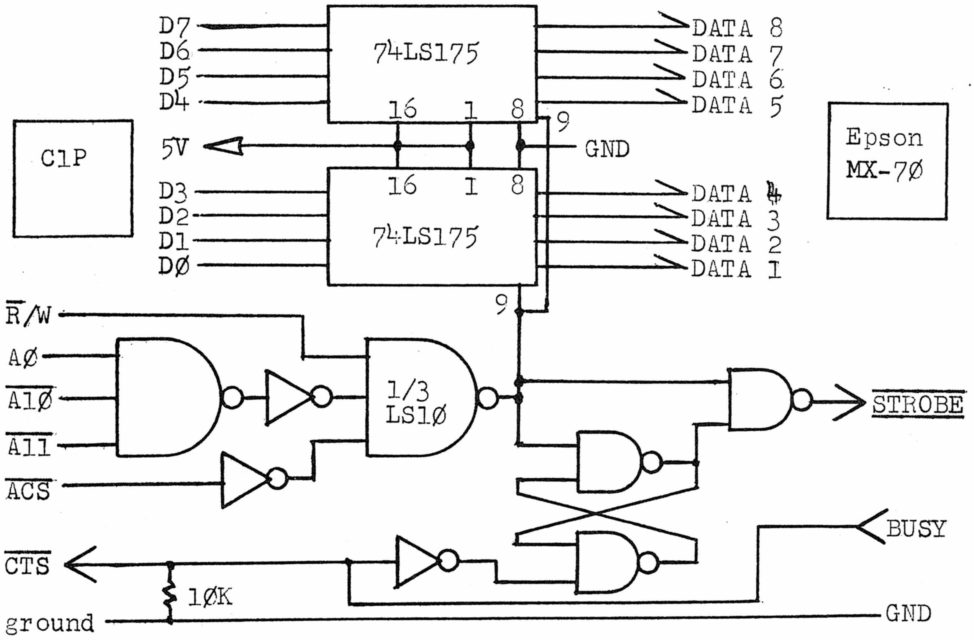
ROY KELLY, N. HOLLYWOOD, CA

I just bough an MX-70 printer for $300, after reading Dec. Aardvark Journal and deciding I could run it with my C1P. I came up with the above modification of Jeff Rae's circuit; after reversing all the data lines because Epson's manual doesn't say which is which, it now works fine as shown below.

To fill the buffer at 9600 buad equivalent with your C1S ROM I found I had to give all the commands in one line as follows:

```
F=61440 : SAVE
POKEF,3:POKEF,16:LIST
```

If you type LIST separately the C1S ROM resets the 6850 back to its divide by 16 mode.



C1P - MX-70 PARALLEL PRINTER INTERFACE
(see also Vol.2 #5 p.6)

KERRY LOURASH, DECATUR, ILLINOIS

The following listing is a correction to the article "CLUTTER for OSI" which was published in the Aardvark Journal Vol. 3 NO. 2 pages 12 and 13. A friend printed the listing and I discovered too late that one line of the program had disappeared. In a ML program, this throws the whole program off; the line cannot just be re-inserted - a whole new assembly must be done.

```
PAGE 001             CLUTTER FOR OSI

     00                        ZPG
     00              LEN       EQU   $17        $3F IF C2/4
     00              LINE      EQU   $20        $40 IF C2/4
     00              ST        EQU   $45
     00              PP        EQU   $47
    0000                       EZP
    0222                       ORG   $222
    0222
    0222 A965                  LDA   #$65       #$40 IF C2/4
    0224 8545                  STA   ST         SET START, POKE POINT
    0226 8547                  STA   PP
    0228 A9D3                  LDA   #$D3       #$D7 IF C2/4
    022A 8546                  STA   ST+1
    022C 8548                  STA   PP+1
    022E D034                  BNE   DECST      BRANCH ALWAYS
    0230
    0230 A2FF        CKLIN     LDX   #$FF
    0232 A0FF        C0        LDY   #$FF
    0234 E8          C1        INX
    0235 C8                    INY
    0236 BD8502                LDA   TBL,X
    0239 F03F                  BEQ   ERASE      IF NULL, ERASE LINE
    023B D145                  CMP   (ST),Y     COMPARE CHAR. TO SCREEN
    023D F0F5                  BEQ   C1         LOOP IF A MATCH
    023F E8          C2        INX
    0240 BD8502                LDA   TBL,X      GET NEXT TBL CHAR.
    0243 D0FA                  BNE   C2         LOOP IF <> 0
    0245 BD8602                LDA   TBL+1,X    DOUBLE NULL?
    0248 D0E8                  BNE   C0         NO, NEXT TBL ENTRY
    024A
    024A A017        PKLIN     LDY   #LEN       LINE LENGTH OF SCREEN
    024C B145                  LDA   (ST),Y     GET CHAR TO BE MOVED
    024E AA                    TAX              SAVE CHARACTER
    024F A920                  LDA   #$20       ERASE OLD CHARACTER
    0251 9145                  STA   (ST),Y
```

```
0253 8A                    TXA              RESTORE CHARACTER
0254 9147                  STA  (PP),Y      PRINT AT NEW LOCATION
0256 88                    DEY
0257 10F3                  BPL  PKLIN+2
0259
0259 38          DECPP     SEC              POKE POINT UP 1 LINE
025A A547                  LDA  PP
025C E920                  SBC  #LINE
025E 8547                  STA  PP
0260 B002                  BCS  DECST
0262 C648                  DEC  PP+1
0264
0264 38          DECST     SEC              START UP 1 LINE
0265 A545                  LDA  ST
0267 E920                  SBC  #LINE
0269 8545                  STA  ST
026B B002                  BCS  DONE
026D C646                  DEC  ST+1
026F
026F A546        DONE      LDA  ST+1        ST < $D000 IF DONE
0271 C9D0                  CMP  #$D0
0273 B0BB                  BCS  CKLIN
0275 A962                  LDA  #$62        POINT MESS. AT $A162
0277 4C78A2                JMP  $A278       TO WARM START
027A
027A A017        ERASE     LDY  #LEN        ERASE A LINE
027C A920                  LDA  #$20
027E 9145                  STA  (ST),Y
0280 88                    DEY
0281 10FB                  BPL  ERASE+4     LOOP IF NOT DONE
0283 30DF                  BMI  DECST       GO TO DECST IF DONE
0285
0285 2020202000 TBL        DATA $20,$20,$20,$20,0
028A 3F                    DATA $3F         "?"
028B 00                    DATA 0
028C 4F4B                  DATA $4F,$4B     "OK"
028E 00                    DATA 0
028F 4C495354              DATA $4C,$49,$53,$54 "LIST"
0293 0000                  DATA 0,0
0295                       END

*****************************
*  S Y M B O L   T A B L E  *
*****************************

C0    =0232  C1    =0234  C2    =023F  CKLIN =0230  DECPP =0259  DECST =0264
DONE  =026F  ERASE =027A  LEN   =0017  LINE  =0020  PKLIN =024A  PP    =0047
ST    =0045  TBL   =0285
```

JIM WEIMER, FT. MORGAN, COLORADO

Have you ever wanted a flag in program that you could easily toggle (like a toggle switch)? Here is a BASIC line which will set the flag ON the first time it is executed, OFF the second time etc.

```
10 LET X = -1         (set flag OFF)
500 X = X - 2*SGN(X)  (toggles flag)
```

X alternates between -1 and 1 by merely executing line 500.

KERRY LOURASH, DECATUR, ILLINOIS
CORRECTION TO BASIC BLOCK DELETE
IN AUGUST VOL. 2 #3 JOURNAL PG. 14

I have now created a much faster version with error checking. It is exactly the same user format. The only difference is that the 'DELETE' flag is now $64. The only way you might get into trouble with the routine (that I can see) is to do an INPUT statement with the output to screen suppressed ($64 set).

VINCE BARBOUR, CINCINATTI, OHIO

Here is a cleaner and perhaps better way to determine if a number is odd or even than using the INT function.

```
ON (TSTNUM and 1)+1 GO TO GOSUB
EVEN ROUTINE 22222, ODD ROUTINE 11111
```

All odd numbers have the low bit=1, all even the the low bit=0. The above expression returns a 0+1=1 for even numbers, a 1+1=2 for odd numbers.

This function can be used to test any bit and can be used handling with the "Secret Basic Functions" in the May 1981 catalog. It is done like this: IF AF AND B1/B8 THEN where B1 though B8 are equal to 2 raised to the first through eighth power.

The THEN branch will be taken if the bit is on. The bit can be set on by using the OR function and B1 through B8.

Each switch takes one bit rather than one byte or more. More space can be saved by using strings and string functions or by using PEEK's, POKE's and free memory.

GLEN FLEISHMAN, EUGENE, OREGON

Below is a sample of an updated version of the routine that will output to a quickprinter without getting strange characters at the beginning of a line because of nulls. The routine that was in Journal #4 (Vol. 1) worked only slightly, so I added a routine that would pause for the length of a null, but not print it, as the other routine did. You still have to poke a value into address 13, but now it has to be about 120 or higher, but it takes less time to execute a null than before. If you change the value that is being loaded into Index register Y (at address $23A), or the amount you POKE into 13, you should be able to get a good speed going with no null problems.

Another thing you need to do with a quickprinter is to install a 600 baud transmit switch. I found an easy way to do it. Take a jumper and wire ti from pin 11, on U30, to one poke of a signal poke, double throw switch (SPDT). Take another jumper and wire it from pin 14 on U59 to the other side of the SPDT. Then, cut the trace leading from pin 2, U57 and solder a wire from the pin to the pole on the switch. The jumpers should be about 6 or more inches long so that you can mount the switch on the back panel, or at least be able to switch it without reaching into the computer. To get 300 baud, put the switch in the position towards the side that has the wire leading from U59. For 600 baud (transmit only), use the other position.

```
ADDR HEX CODE MNEMONIC

0000 4CD800    JMP $00D8
00D8 A922      LDA #$22
00DA 8D1A02    STA $021A
00DD A902      LDA #2
00DF 8D1B02    STA $021B
00E2 4C74A2    JMP $A274
------------------------------
0222 202DBF    JSR $BF2D
0225 48        PHA
0226 AD0502    LDA $0205
0229 D002      BNE $022D
022B 68        PLA
022C 60        RTS
022D 68        PLA
022E C900      CMP #0
0230 F003      BEQ $0235
0232 4CB1FC    JMP $FCB1
0235 48        PHA
0236 8A        TXA
0237 48        PHA
0238 98        TYA
0239 48        PHA
023A A005      LDY #5
023C A2FF      LDX #$FF
023E CA        DEX
023F D0FD      BNE $023E
0241 88        DEY
0242 D0F8      BNE $023C
0244 68        PLA
0245 A8        TAY
0246 68        PLA
0247 AA        TAX
0248 68        PLA
0249 60        RTS
```

FOR SALE: C2-4P MOD 3 (C4P WITH COLOR UNPOPULATED). 16K (SOCKETED FOR 32K), PRINTER PORT, XTRA 5V 6A POWER SUPPLY, STRING BUG FIX #3 ROM, $350 IN SOFTWARE BEST OFFER. STEVEN GALE (216) 752-4845

FOR SALE: C2/4P 32K IN EXCEL. SHAPE, PLENTY DOCUM., SOFTWARES & EXTRAS. $475 OR OFFER, (AL) 414-278-0428, 710 E. MASON #75, MILWAUKEE, WI 53202

JOHN C. SCHERR, PETERSBURG, VIRGINIA

After seeing the Animal Guess game for disk in your April 82 issue I thought the C1P users might feel left out. This should work on other systems as well, with maybe a change in the WAIT command.

ANIMAL GUESS

```
 10 REM ANIMAL GUESS
 20 REM  AUTHOR UNKNOWN - MODIFIED BY JO
HN SCHERR FOR
 30 REM  INPUT & OUTPUT TO CASSETTE BASE
D FILE
 40 FORCS=1TO32:PRINT:NEXT
 50 DEFFNA(I)=I-N:CLEAR:DIMQ$(50),A1$(50
),L1(50),L2(50)
 60 INPUT"Does the data file exist";A$
 70 IFA$<>"Y"THENQ$(1)="DOES IT HAVE 4 F
EET":L1(1)=1:L2(1)=999:N=1
 80 IFA$<>"Y"THENA1$(1)="HORSE":GOTO170
 90 PRINT:PRINT"Play data tape.  Press S
HIFT when tone starts.
 100 WAIT57088,254,254:LOAD
 110 I=I+1:INPUTQ$(I),L1(I),A1$(I),L2(I)
 120 IFQ$(I)<>"END"THEN110
 130 POKE515,0:FORCS=1TO32:PRINT:NEXT
 140 PRINT:PRINT"DATA ENTERED.":PRINT
 150 N=I-1:INPUT"Continue ";A$:PRINT:PRI
NT:PRINT
 160 IFN>50THENPRINT"C A R E F U L ! MOR
E THAN 50!"
 170 PRINT:PRINT:PRINT:INPUT"ARE YOU THI
NKING OF AN  ANIMAL";A$
 180 IFA$="L"THENK=1
 190 IFA$="Y"THENK=2
 200 IFA$="D"THENK=3
 210 IFA$="S"THENK=4
 220 ONKGOSUB250,280,480,510
 230 PRINT:PRINT
 240 INPUT"Y(ES), L(IST), S(AVE),  OR D(
EBUG) ";A$:PRINT:PRINT:GOTO180
 250 PRINT:PRINT"THE ANIMALS I KNOW ARE
...
 260 FORI=1TON:PRINTA1$(I),:NEXTI
 270 INPUT"Continue ";A$:PRINT:PRINT:PRI
NT:RETURN
 280 I=1:REM "Y"
 290 PRINT:PRINTQ$(I);:INPUTA$:PRINT
 300 IFA$="Y"THEN350
 310 IFA$<>"N"THEN290
 320 IFL2(I)<>999THENI=L2(I):GOTO290
 330 GOSUB440
 340 L2(I)=N+1:GOSUB470:RETURN
 350 PRINT"IS IT ";A1$(I);:INPUTA$
 360 PRINT
 370 IFA$<>"Y"THEN400
 380 PRINT:PRINT"...I THOUGHT SO."
 390 FORT=1TO4000:NEXT:RETURN
 400 IFA$<>"N"THEN350
 410 IFI<>L1(I)THENI=L1(I):GOTO290
 420 GOSUB440
 430 L1(I)=N+1:GOSUB470:RETURN
 440 INPUT"WHAT WAS THE ANIMAL YOU WERE
THINKING OF          ";A$
 450 PRINT:PRINT"TYPE A QUESTION THAT WO
ULD DISTINGUISH
 460 PRINTA$;" FROM "A1$(I):INPUTQ1$:RET
URN
 470 N=N+1:Q$(N)=Q1$:A1$(N)=A$:L1(N)=N:L
2(N)=999:RETURN
 480 PRINT:PRINT"I    Q$(I)           A1$(I
)       L1(I)        L2(I)
 490 FORI=1TON:PRINTI;Q$(I);"    ";A1$(I)
;"   ";L1(I);"   ";L2(I):NEXTI
 500 PRINT:INPUT"Continue ";A$:RETURN
 510 REM SAVE ROUTINE
 520 PRINT:PRINT"Set up to record data t
ape.  Press SHIFT when ready.
 530 WAIT57088,254,254:POKE517,1
 540 FORI=1TON
 550 PRINTQ$(I);",";L1(I);",";A1$(I);","
;L2(I)
 560 NEXTI
 570 PRINT"END,"O",END,";O
 580 FORT=1TO500:NEXT:POKE517,0
 590 PRINT:PRINT"DATA STORED":PRINT
 600 INPUT"Continue";A$:RETURN
```

GALAXIA ADDITION

ROBERT CORWIN, PORTVILLE, NEW YORK

Recently I was working on one of Aardvark's check sum programs namely "Galaxia" trying to find a way to get the score and other information that on the Superboard is printed off screen to print where I would be able to see it.

I not only found how but I found a way to make a hard copy at the

First set up the program per instructions and hit the BREAK key- Hit M- and Enter 08A3/64 * 08A4/DO * 0886/70 * 0887/DO * 08C0/71 * 08C1/DO * 08CC/72 * 08CD/DO * 086D/73 * 08D7/DO * 08D8/74 * 08DC/DO * 08ED/78 * 08EE/DO * 08FD/69 * 08FE/DO * 08E3/69 * 08E4/DO

This will put the information at the top of the screen above the game and will not interfere with the game.

To change the tanks to something different GOTO 02AD and insert A-04 and at 0281 insert A-04 this will put small flying saucers in place of the tanks.

"! TO RESTART THE GAME ENTER .0250G AND THE GAME WILL RESTART.

To make a hard copy of this or any check sum program do the following: First Cold start the computer and then enter the following two lines in BASIC:

```
1 POKE515,255:DIMA$(150),B$(150):
FORX=1TO150:INPUTA$(X):NEXT
2 FORX=1TO150:INPUTB$(X):NEXT
```

When the above is entered set up your Galaxia tape and start the recorder and type RUN, when the program is in hit the space bar and the Return key to return to keyboard opp.

At this time "ALL" entries must be made direct with "NO" line numbers or you will wipe out all of your strings and have to start over again.

Remember that the first 225 bits are the check sum operator, so unless you want a copy of this as well as the program advance your tape to where the long lines of code list to the screen and then RUN.

Now start your printer or teletype as the case may be and enter:

```
FORX=1TO150:?A$(X):NEXT:FORX=1TO150:?B$(X):
NEXT--AND HIT RETURN.
```

If you want the check sum operator also, then install A; after the Print loop so as not to have long line of single entries as the first 225 bits are in single entry format.

ALIEN II FOR THE C1E CHIP

by NELSON REYNOLDS, MICHIGAN

```
20 FORSH=590TO673:READY1:POKESH,Y1:NEXT
30 POKE11,78:POKE12,2
250 12=L1*2:POKE657,12
910 IFPEEK(252)=0THENGOSUB1130
1130 IP=IP+L1:TV=IP:AF=0:DI=1:
POKEAP,32:SM=SM-1:IFTU<=3THEN420
```

Lines 20,30 and 250 reset the USR routine to unused space. Line 910 is the line that was hitting the return error in line 1260. Line 1130 keeps the computer from getting confused when TU=3. By the way, the corrected portions are underlined in the above program lines. Also, so far I have not been able to find any cross between TV and TU.

PING PONG GAME, by John Seybold

Here is a good (2) man Ping Pong game. It was written on my Superboard (series I) and will run in 4K. I did not know how to make it work on any of the bigger machines so I will leave it up to you. I tried in most cases to use variable names that make sense such as BL for bottom left of the screen, LP for the left paddle character etc... Some acceptions are; L for difficulty level, LS was the starting address of the left paddle, but is also the address of the paddle after it has been moved around, I also reused SL$ for the play again?

```
 10 REM-PING PONG BY JOHN S. SEYBOLD
 11 REM-1322 BROAD ST.
 12 REM-OSHKOSH, WI 54901
 20 GOSUB1000:GOTO500:ADD SOUND TO LINE
500
 29 REM-Z IS THE DIRECTION OF THE SERVE;
 3-LEFT, 0-RIGHT
 30 D=DI(INT(3*RND(2)+1+Z)):AD=CE-3:IFZT
HENAD=CE+3
 99 REM-PROGRAM LOOP
 100 POKEAD,BA:Y=PEEK(AD+D):IFY<>TWTHEN1
20
 105 IFD=-33THEND=31:GOTO200
 110 IFD=-31THEND=33:GOTO200
 120 IFY<>BWTHEN140
 125 IFD=33THEND=-31:GOTO200
 130 IFD=31THEND=-33:GOTO200
 140 IFY=RWTHENS1=S1+1:Z=0:POKEAD,V:GOTO
500
 150 IFY=LWTHENS2=S2+1:Z=3:POKEAD,V:GOTO
500
 160 IF(Y<>LP)AND(Y<>RP)THEN180
 165 IFD=31THEND=33:GOTO200
 170 IFD=33THEND=31:GOTO200
 175 IFD=-31THEND=-33:GOTO200
 177 IFD=-33THEND=-31:GOTO200
 180 IFPEEK(AD+1)=RPTHEND=DI(INT(RND(5)*
3+4)):GOTO200
 190 IFPEEK(AD-1)=LPTHEND=DI(INT(RND(6)*
3+1))
 200 POKEAD,V:AD=AD+D:POKEAD,BA
 299 REM-READ KEYBOARD
 300 POKEKB,253:Y=PEEK(KB):IF(YOR127)<>1
27THEN320
 310 POKELS,V:LS=LS-V:IFPEEK(LS)<>VTHENL
S=LS+V
 320 IF(YOR191)<>191THEN330
 325 POKELS,V:LS=LS+V:IFPEEK(LS)<>VTHENL
S=LS-V
 330 IF(YOR253)<>253THEN340
 335 POKERS,V:RS=RS-V:IFPEEK(RS)<>VTHENR
S=RS+V
 340 IF(YOR251)<>251THEN350
 345 POKERS,V:RS=RS+V:IFPEEK(RS)<>VTHENR
S=RS-V
 350 POKELS,LP:POKERS,RP:FORX=1TO100-10*
L:NEXT:GOTO100:PROGRAM LOOP
 500 FORX=26TO43:POKEAD,X:NEXT:S1$=STR$(
S1)+"  ":S2$=STR$(S2)+"  "
 510 POKEP1+1,ASC(RIGHT$(S1$,2)):POKEP1,
ASC(RIGHT$(S1$,3))
 520 POKEP2+1,ASC(RIGHT$(S2$,2)):POKEP2,
ASC(RIGHT$(S2$,3))
 530 POKEAD,V:IFS1<21ANDS2<21THEN30
 540 PRINT:INPUT"PLAY AGAIN";S1$:IFASC(S
1$)=89THEN20
 550 POKEP1,V:POKEP1+1,V:POKEP2,V:POKEP2
+1,V
 560 FORX=1TO30:PRINT:NEXTX:POKE530,0:EN
D
 999 REM-INTRODUCTION
 1000 FORX=1TO10:PRINT:NEXTX:PRINT"
  PING PONG"
 1010 FORX=1TO6:PRINT:NEXTX:PRINT"LEFT M
AN USES  'Q' & 'A'"
 1020 PRINT:PRINT"RIGHT MAN USES 'P' & '
;'":PRINT
 1030 PRINT"TO CONTROL PADDLES":PRINT:PR
INT:PRINT
 1040 INPUT"DIFFICULTY (1-10)";L:IF(L<1)
OR(L>10)THEN1040
 1050 FORX=1TO30:PRINT:NEXTX
 1099 REM-INITIALIZATION
 1100 FORX=1TO6:READDI(X):NEXTX:RESTORE:
POKE530,1
 1200 BL=54118:WIDTH=22:HEIGHT=18:V=32:T
W=154:BW=155:LW=157:RW=156
 1210 LP=153:RP=152:LS=53831:RS=LS+WI-2:
CE=(LS+RS)/2:BALL=226
 1220 KB=57088:FORX=BLTOBL+WI:POKEX,BW:N
EXTX
 1240 FORX=BLTOBL+WI:POKEX,BW:NEXTX
 1250 FORX=BL+WI-V*HETOBL-V*HESTEP-1:POK
EX,TW:NEXTX
 1260 FORX=BL+WITOBL+WI-V*HESTEP-V:POKEX
,RW:NEXTX
 1270 FORX=BL-V*HETOBLSTEPV:POKEX,LW:NEX
TX
 1280 POKELS,LP:POKERS,RP:Z=0:IFRND(6)>.
5THENZ=3
 1290 P1=LS+V*11:P2=P1+18:AD=CE:S1=0:S2=
0:RETURN
 1300 DATA33,1,-31,-33,-1,31
```

ABOUT POOL (2 PLAYERS) by D.L. Davis

NOTE FROM EDITOR: This game may need some adjustments which are up to you. Here is the letter from the author which may help you.

It was structured sometime ago and if I was to start over I'd probably use a PRINT AT statement for scoring and prompts etc., but as the string bug subtracts 16 from the PRINT AT (unless mem is moved) I'll leave it as is.

This one has a break that works pretty well. The balls could be moved farther on the break, but too many may end up against the rail unless I add a routine.

Having the arrows and the numbers of keys is a distraction, at least to me. While playing the game I just draw a small direction chart to use. The C counter works with the internal times to slow the ball down so the shot diminishes, and it also limits the count to 50 when a rolling ball hits a cueball. This is the only bug and it happens rarely when the rolling ball pushes the cueball against the rail.

After the ball is made, a # between 1 and 15 is assigned to it and not reused making the score total =120 and permitting a tie game. The ball made indicator is in the lower right of CRT.

I use U and V a lot for determining which player, and there may still be some wasteful lines in there though I took some out.

```
POOL VARIABLE LIST
RD R1 R2 R8 R RANDOM #'S
G1 G2 G3 LEFT PLAYER SCORE LOC.
J1 J2 J3 RIGHT " " " " " " " "
U V        PLAYERS TURN INDICATOR LOC.
D          DIRECTION OF BALL TRAVEL
L1 L2      LOC. OF BALL MADE INDICATOR
P1 TO P6 POCKETS
X          LOC OF ROLLING BALL
HX         HOLD LOC.OF BALL THE HITS BALL
LO W() V() DO() LOC. THE BREAK
C          COUNTER FOR DURATION OF ROLL
P          LOC. AHEAD OF ROLLING BALL
E          EDGES OF TABLE
EE         LAST DIRECTION
N,S,NE,W,ETC.
A(1) TO A(8)  DIRECTIONS
Z() LIST OF BALLS MADE
SUBROUTINES
4   SCREEN CLEAR
300 CHANGE PLAYERS
400 SCRATCH
500 DIR OF SHOT CHOSEN
600 POKE TABLE AND POCKETS
700 POKE PLAYER + 000
800 SCORING
1000 DIR. ARROWS PER KEYS 1-7
2000 INSTRUCTIONS
```

```
 1 REM REV 5.5 POOL COPYRIGHT BY D.L. DA
VIS
 2 REM FT. WAYNE INDIANA
 3 GOTO11
 4 A=PEEK(129):B=PEEK(130):POKE129,0:POK
E130,212:S$=" ":FORSS=1TO7
 5 S$=S$+S$+" ":NEXT:POKE129,A:POKE130,B
:RETURN
 11 LO=53452:DIMW(15):DIMV(15):DIMDD(15)
 12 PRINT"INSTRUCTIONS NEEDED?":INPUTI$:
IFLEFT$(I$,1)="Y"THEN2000
 13 GOSUB4:RD=INT(RND(1)*99+1)
 14 F1=53896:G2=53894:G3=53893:J1=53916:
J2=53915:J3=53914:G1=53895
```

```
 15 IFI$="Y"THENGOS[illegible]1000
 16 U=53830:V=53851:F=54221:B=226:C=0:D=
N:BA=111
 18 POKE53617,B:POKE53584,B:POKE53586,B:
POKE53551,B:POKE53553,B
 19 POKE53555,B:POKE53518,B:POKE53520,B:
POKE53522,B:POKE53524,B
 20 POKE53485,B:POKE53487,B:POKE53489,B:
POKE53491,B:POKE53493,B
 23 N=-32:S=+32:EE=+1:W=-1:NE=-31:NW=-33
:SW=+31:SE=+33
 24 A(1)=+1:A(2)=-31:A(3)=-32:A(4)=-33:A
(5)=-1:A(6)=+31
 25 A(7)=+32:A(8)=+33
 26 P1=53353:P2=53769:P3=54185:P4=53369:
P5=53785:P6=54201
 28 L1=54202:L2=54203:DIMZ(15)
 34 GOSUB600:GOSUB700:X=54033
 38 FORLO=LOTO53679:IFPEEK(LO)=226THEN40
 39 IFPEEK(LO)<>226THEN45
 40 R8=INT(RND(7)*8+1)
 41 CU=CU+1:W(CU)=LO:V(CU)=R8
 45 NEXT
 46 FORXX=XTOX-380STEP-32:POKEXX,111:FOR
T=0TO50:NEXT:POKEXX,32
 47 NEXT
 48 X=XX:POKEX,111
 49 GOSUB730
 50 D=A(R8)
 52 R=INT(RND(RD)*10+1):GOTO62
 61 X=HX
 62 BA=111
 63 R2=INT(RND(RD)*8+1)
 65 FORY=54221TO54229:POKEY,32:NEXT
 70 R=INT(RND(C)*8+1)
 71 C=C+1
 73 P=PEEK(X+D):IFP<>32THEN80
 76 FORT=0TOC:NEXT:POKEX,32
 77 X=X+D:POKEX,BA
 78 IFC>100THENGOSUB300:GOTO500
 79 DT=DT+1:GOTO70
 80 IFP=157ANDD=NWTHEND=EE:GOTO70
 81 IFP=156ANDD=EETHEND=NW:GOTO70
 82 IFP=154THEND=SW:GOTO70
 83 IFP=155THEND=NE:GOTO70
 84 IFP=156ANDD=SETHEND=W:GOTO70
 85 IFP=156ANDD=NETHEND=NW:GOTO70
 86 IFP=157ANDD=SWTHEND=SE:GOTO70
 87 IFP=157THEND=SE:GOTO70
 93 IFP=226THEN200
 94 IFP=96THENPOKEX,32:GOTO400
 98 IFP=111THENGOSUB100
 99 GOTO70
 100 REM
 110 IFPEEK(HX)=111THEN120
 115 RETURN
 120 IFPEEK(HX+A(R2))=32THEN130
 121 IFR2<8THENR2=8:GOTO120
 122 IFR2>1THENR2=1
 123 C=C+1
 127 IFC>050THENGOSUB300:GOTO500
 128 GOTO120
 130 POKEHX,32:HX=HX+A(R2):POKEHX,111
 140 RETURN
 200 REM
 201 IFPEEK(X)=111THENHX=X
 202 BA=226
 205 X=X+D
 206 R=INT(RND(RD)*8+1):D=A(R)
 210 IFPEEK(X+D)<>32THEN206
 215 POKEX,32:GOTO70
 300 REM
 310 IFPEEK(U)=49THENPOKEU,32:POKEV,50:G
OTO350
 340 POKEV,32:POKEU,49
 350 RETURN
 400 X=X+D:POKEX,BA::FPEEK(X)=226THENGOS
UB800:GOTO490
 401 POKEX,96:HX=0:X=54097
 402 IFPEEK(X)=226THENX=X+1:GOTO402
 405 POKEX,111
 406 POKEF,83:POKEF+1,67:POKEF+2,82:POKE
F+3,65:POKEF+4,84
 407 POKEF+5,67:POKEF+6,72
 411 IFPEEK(U)=49THEN413
 412 IFPEEK(V)=50THEN414
 413 POKEU,32:POKEV,50:POKEV+32,145:GOTO
415
 414 POKEV,32:POKEU,49:POKEU+32,145
 415 POKE530,1:K=57088:POKEK,191
 420 P=PEEK(K)
 422 IFP=127THEN440
 423 IFP=223THEN450
 424 IFP=191THEN500
 435 POKE530,0:GOTO415
 440 IFX<54092THENX=54091:GOTO415
 443 IFPEEK(X-1)=226THEN415
 444 POKEX,32:FORT=0TO20:NEXT:X=X-1:POKE
X,111
 445 GOTO415
 450 IFX>54102THENX=54103:GOTO415
 454 IFPEEK(X+1)=226THEN415
 459 POKEX,32:FORT=0TO20:NEXT:X=X+1:POKE
X,111
 460 GOTO415
 490 REM
 498 POKEX,96
 500 IFPEEK(X)=111THENHX=X
 501 IFPEEK(U)=49THENPOKEU+32,155
 502 IFPEEK(V)=50THENPOKEV+32,155
 503 R=INT(RND(RD)*8+1):C=0
 505 POKE530,1:K=57088:POKEK,127:P=PEEK(
K):D=0
 508 POKEF,68:POKEF+1,73:POKEF+2,82:POKE
F+3,69:POKEF+4,67
 509 POKEF+5,84:POKEF+6,73:POKEF+7,79:PO
KEF+8,78
 516 IFP=(127AND191)THEND=S
 518 IFP=127THEND=N
 520 IFP=191THEND=NW
 530 IFP=223THEND=W
 540 IFP=239THEND=SW
 550 IFP=247THEND=SE
 560 IFP=251THEND=EE
 570 IFP=253THEND=NE
 571 POKEU+32,32:POKEV+32,32
 577 IFD<>0ANDHX=0THEN62
 579 IFD<>0THENPOKEL1,32:POKEL2,32
 580 IFDTHEN61
 590 POKE530,0:GOTO505
 600 REM
 629 FORE=P1TOP3STEP32:POKEE,157:NEXT
 630 FORE=P4TOP6STEP32:POKEE,156:NEXT
 631 FORE=P1TOP4:POKEE,154:NEXT
 632 FORE=P3TOP6:POKEE,155:NEXT
 633 POKEP1,96:POKEP2,96:POKEP3,96:POKEP
4,96:POKEP5,96:POKEP6,96
 634 POKEP1+32,96:POKEP3-32,96:POKEP4+32
,96:POKEP6-32,96
 635 POKEP1+1,96:POKEP4-1,96:POKEP3+1,96
:POKEP6-1,96
 636 POKEP2+1,96:POKEP5-1,96
 640 RETURN
 700 REM
 701 POKEU,49
 702 POKEG1,48:POKEJ1,48:POKEG2,48:POKEG
3,48:POKEJ2,48:POKEJ3,48
```

```
 703 POKE53627,80:POKE53659,76:POKE53691
,65:POKE53723,89:POKE53755,69
 704 POKE53787,82
 710 POKE53606,80:POKE53638,76:POKE53670
,65:POKE53702,89
 711 POKE53734,69:POKE53766,82
 720 RETURN
 730 Y=15
 733 Y=Y-1:DD(Y)=A(V(Y)):PP=W(Y)+DD(Y)
 734 IFPEEK(PP)<>32THEN741
 735 IFPEEK(W(Y))=226THENPOKEW(Y),32
 736 IFPEEK(PP)=32THENPOKEPP,226
 741 IFY=1THENRETURN
 742 GOTO733
 800 REM
 801 R1=INT(RND(RD)*15+1)
 802 FORZ1=1TO15
 804 IFZ(Z1)=R1THEN801
 806 IFZ1=R1THENZ(Z1)=R1:GOTO809
 808 NEXT
 809 IFZ1<10THENPOKEL1,32:POKEL2,Z1+48:G
OTO811
 810 POKEL1,49:POKEL2,(Z1-10)+48
 811 IFPEEK(U)=49THEN820
 814 IFPEEK(V)=50THEN850
 820 SL=SL+R1
 822 S6=INT(SL/100):S5=INT(SL/10):S4=SL-
(INT(SL/10)*10)
 828 IFS5>9THENS5=S5-10
 845 POKEG3,S6+48:POKEG2,S5+48:POKEG1,S4
+48
 846 IFSL>60THENGOSUB4:PRINT"PLAYER ONE
WINS"SL" TO "SR:END
 848 IFSL+SR=120THENGOSUB4:PRINT"TIE GAM
E":END
 849 RETURN
 850 SR=SR+R1
```

```
 852 S3=INT(SR/100):S2=INT(SR/10):S1=SR-
(INT(SR/10)*10)
 858 IFS2>9THENS2=S2-10
 895 POKEJ3,S3+48:POKEJ2,S2+48:POKEJ1,S1
+48
 896 IFSR>60THENGOSUB4:PRINT"PLAYER TWO
WINS"SR" TO "SL:END
 898 IFSL+SR=120THEN848
 899 RETURN
 1000 I=23:FORY=53350TO53542STEP32:IFI=2
0THENI=19
 1020 POKEY,I:I=I-1:NEXT:I=50
 1040 FORY=53352TO53544STEP32:IFI=56THEN
I=49
 1050 POKEY,I:I=I+1:NEXT:POKE53991,20
 1071 POKE54054,49:POKE54055,38:POKE5405
6,50:RETURN
 2000 PRINT"PUSH NUMBERS AS SHOWN":PRINT
 2010 PRINT"ON CRT TO SHOOT IN":PRINT
 2020 PRINT"DIRECTION OF ARROW":PRINT:PR
INT
 2030 PRINT"PUSH 8 AND 0 TO MOVE":PRINT
 2040 PRINT"CUEBALL AFTER A SCRATCH":PRI
NT:PRINT
 2050 PRINT"PUSH 9 TO PREPARE TO":PRINT
 2060 PRINT"SHOOT AFTER A SCRATCH":PRINT
:PRINT
 2070 PRINT"FIRST ONE TO GET 61 WINS"
 2072 PRINT"PUSH 1 AND RET TO SHOOT FIRS
T SHOT"
 2077 INPUTA
 2080 IFA=1THEN13
 2085 GOTO2000
```

AARDVARK
TECHNICAL SERVICES
2352 South Commerce
Walled Lake, MI 48088

ROYAL OAK, MI
PM
AUG 16
1982

Rachel Carson
USA 17c

USA 20c

J. Cook
402 N. Shore Dr.
Bellingham, WA.
98226

FIRST CLASS